Status of the ACTS Project
A Common Tracking Software

A. Salzburger (CERN) on behalf of the ACTS team
Motivation and Mission statement

- LHC Run-1/2 exceeded all expectations in terms of provided data
  - design pile-up of $<\mu> \sim 23$ exceed (for Run-2 de facto with a factor 2)
  - trackers and track reconstruction worked extremely well

- LHC track reconstruction software applications
  - extremely well tested code ($10^{10}$ events with up to $10^3$ tracks/event)
  - extremely high performant code

- However, ...
  - code base and concepts are <fill in your favourite word for ageing>
  - HL-LHC and future FCC will bring even more demanding
  - developments in tracker, computing hardware and algorithmic concepts
A ... embedded in Gaudi/Athena structure with AlgTools/Algorithms/Services
## Tracking

<table>
<thead>
<tr>
<th>Geometry (TrkDetDescr)</th>
<th>Event Data Model (TrkEvent)</th>
<th>Extrapolation (TrkExtrapolation)</th>
<th>G/A</th>
<th>Fitting (TrkFitter)</th>
<th>G/A</th>
<th>Calibration, general (TrkTools)</th>
<th>G/A</th>
<th>Alignment (TrkAlignment)</th>
<th>G/A</th>
</tr>
</thead>
</table>

## a-tracking-sw

Attempt to encapsulate the ATLAS Tracking software from ATLAS

**currently restricted access**

- **Star:** 0
- **Push:** 1
- **SSH**
- **HTTPS**

```
ssh://git@gitlab.cern.ch:7999/
```

- **Commits:** 65
- **Branches:** 5
- **Tags:** 0
- **Size:** 13.21 MB

## A Tracking Software (ATS) Project

This library is based on the track reconstruction software developed by the ATLAS Collaboration.

The main philosophy is to provide high-level track reconstruction modules that can be specified for detector technologies by simple extension.

- **Event Data Model (EDM)**
- **Geometry**
- **track reconstruction tools**

The library is attempted to build against **Gaudi** and **Gaudi-Athena**, while additional external dependencies are kept at a minimum.

**double build philosophy**
Established an open-source ACTS project at CERN/gitlab
- growing functionality
- growing developers base
  mainly from ATLAS and FCCSW
- main clients are
  FCCSW (mainly for FCC-hh)
  Tracking ML challenge (starting)
  ATLAS
- Latest release 0.03.01,
  in synch with FCC development

http://acts.web.cern.ch/ACTS/
https://gitlab.cern.ch/acts/a-common-tracking-sw
https://its.cern.ch/jira/projects/ACTS/summary
Dependencies, Repository and Modules

**a-common-tracking-sw**

**Core**
- Event Data Model
- Geometry & building
- Transport
- Fitting
- Pattern recognition

**Plugins**
- DD4HepPlugin
- TGeoPlugin

---

**Build infrastructure**
- gcc (>= 6.2 ) or clang (>= 3.8 )
- cmake (>= 3.5 )

**Core Dependencies**
- eigen (linear algebra)
- boost (unit testing)

**Plugin Dependencies (optional)**
- DD4hep
- ROOT (> 6.0)

---

**acts-mini-fw**

**Core**

**Plugins**

---

**acts-fatras (not yet public)**

**Core**

**Plugins**

---

- Geant4 for material mapping
- ROOT for writing

- Geant4 for hadronic interactions

---

docker images available
Testing, testing, testing

- Aim is to provide long-term maintainable code/toolbox
  - testing is crucial

- Continuous integration testing using Jenkins for every merge request
  - CI success is one required approver for accepting a merge request
  - Human review is the second required approval
  - aim for minimal disruption of integration

- Unit test suite is growing, but certainly not yet optimal …
  - based on boost test framework
Documentation & code style

- Aim is to provide long-term maintainable code/toolbox
  - documentation is crucial, consistent code-style is essential

- doxygen for code documentation
  - one central documentation source
  - contribution guide gives guidelines for doxygen usage
  - code is formatted to clang-format by CI system

Detailed Description

Class for material mapping.

This class should be used to map material from the full and detailed detector geometry onto the simplified ACTS geometry. It offers options to map, average and finalize the material. One `MaterialTrackRecord` (containing all the MaterialSteps along a Track) is mapped by using the function `Acts::MaterialMapping::mapMaterial()`. The mapping process then extrapolates into the same direction starting from the same point as the `MaterialTrackRecord` through the ACTS geometry and finds the closest surface of a layer which is marked to carry support material. The material is assigned to the closest layer (forward or backward) to the bin at the assigned position on the layer. Along one track in one bin of a layer the material is averaged:

\[
\frac{t}{x_0} = \sum_{i=1}^{n} \frac{t_i}{x_i}, \quad \frac{t}{\lambda_0} = \sum_{i=1}^{n} \frac{t_i}{\lambda_i}, \quad \rho = \frac{\sum_{i=1}^{n} t_i \rho_i}{\sum_{i=1}^{n} t_i},
\]

\[
A = \frac{\sum_{i=1}^{n} \rho_i A_i}{\sum_{i=1}^{n} \rho_i}, \quad Z = \frac{\sum_{i=1}^{n} \rho_i Z_i}{\sum_{i=1}^{n} \rho_i}, \quad\text{sensper} = \frac{\sum_{i=1}^{n} \text{sens}V_i}{\sum_{i=1}^{n} V_i}.
\]

\( t \)...thickness, \( x_0 \)...radiation length, \( \lambda_0 \)...interaction length, \( \text{sensper} \)...sensitive percentage, \( V \)...Volume, \( \rho \)...density, \( A \)...mass number, \( Z \)...atomic number

Pillars of track reconstruction

- **Geometry**
  - ACTS provides a plugin algorithm for different geometry backends
  - helper tools in place to translate this into ACTS geometry
  - automated material transcript from full (Geant4) detector geometry

- **Event Data Model**
  - ACTS provides a standard implementation for track parameters
    parameterisation can be changed if needed
  - Measurement description does not restrict you
    1-N dimensional measurements

- **Algorithms**
  - ACTS provides Extrapolation/Propagation through arbitrary magnetic field
    optimised for ACTS detector geometry
  - Track fitters, pattern recognition algorithms, fast simulation are in preparation
ACTS builds tracking geometry model from detailed geometry input

```cpp
namespace Acts {
    /// doxygen documentation
    class DetectorElementBase {
        /// the according represented surface
        virtual const Surface& associatedSurface() const = 0;
    };
}

class MyDetectorElement {
    /// @copydoc DetectorElementBase::associatedSurface
    const PlaneSurface& associatedSurface() const;
};
```

![Diagram](image)
Geometry Examples - DD4Hep

- Different detectors have been built in ACTS with DD4Hep input
  - FCC-hh detector
  - CLIC detector example
  - ATLAS IBL description

- A dedicated DD4Hep/TGeo plugin provided within ACTS
  - DD4Hep uses TGeo in the background for as a modeller (also TkLayout support)
  - common TGeoPlugin used for DD4Hep geometry backend and TGeo backend

```cpp
std::unique_ptr<Acts::TrackingGeometry>
convertDD4hepDetector(DD4hep::Geometry::DetElement worldDetElement,
    Logging::Level loggingLevel = Logging::Level::INFO,
    BinningType bTypePhi = equidistant,
    BinningType bTypeR = equidistant,
    BinningType bTypeZ = equidistant,
    double layerEnvelopeR = 0.,
    double layerEnvelopeZ = 0.);
```
Geometry Examples - FCC studies

- FCC-hh tracker design workflow
  - starting with TkLayout tool (CMS)
  - translation into TGeo (DD4Hep) -> ACTS
Material description & mapping

- Tracker needs an appropriate material description
  - usually simplified version of full simulation description
  - often used as material description for fast simulation

- Any geometry backend that also creates a full Geant4 description
  - automated material mapping procedure in place
  - uses Geant4 detector material and maps it onto chosen ACTS geometry structure
Material mapping - examples

- All ACTS surfaces can carry material
  - mapping targets to be chosen
  - 0,1,2 dimensional binning

- Alternatively material maps can be also provided externally
  - e.g. through direct transcript
Event Data Model - track parameterisation

- Fixed-size track parameterisation
  - size can by chosen at compile time as a template parameter `NGlobalPar`
    (e.g. extend with time, see talk of L. Gray)
  - default parameterisation follows ATLAS choice
    all of the derivates needed are available & tested
  - parameterisation can be chosen differently, if wanted:
    Jacobian matrices to curvilinear need to be provided
    measurement mapping functions need to be provided

\[
\begin{align*}
\text{CDF} & \quad q'' = (l_1, l_2, \phi, \cot(\theta), C') \\
\text{CMS} & \quad q' = (l_1, l_2, \phi, \lambda, q/p) \\
\text{ATLAS} & \quad q = (l_1, l_2, \phi, \theta, q/p)
\end{align*}
\]
Event Data Model - measurements

- Measurements are dynamic-sized
  - possibility to provide each subset of $1-N_{\text{GlobalPar}}$ dimensional measurements
  - for an ideal/most performant choice:
    measurement mapping functions become projection matrices

```cpp
ParameterSet<loc1, phi> pl(0.5, 1.5*M_PI);
 cout << pl->get<loc1>() << endl; // prints 0.5
 cout << pl->get<loc2>() << endl; // gives compiler error
 cout << pl->get<phi>() << endl; // gives -0.5*M_PI
 pl->getParameters(); // returns fixed size vector
 pl->projector(); // returns fixed size projection matrix
```
Propagation and Extrapolation

- Extrapolation engine design from ATLAS Tracking SW
  - splits mathematical propagation from material effects integration and navigation
  - allows to plug in:
    - different propagators
    - different material effects integration

ExtrapolationCell

StaticEngine

DynamicEngine

DenseEngine

configuration, cache for material, parameters, jacobians
Extrapolation - geometry based

for fully connective geometry and layer configuration

for finding “floating objects”

for transport through dense material, using a dedicated STEP propagator
Fast digitisation

- Geometric digitisation
  - first imported module from ATLAS fast track simulation FATRAS
  - calculates cluster sizes and path lengths in individual pixels
    validated against ATLAS Geant4 simulation
  - takes Lorentz angle into account
  - allows for smearing of charge deposit

cluster shapes for tracks in Machine Learning dataset (see talk of S. Amrouche)
Track finding - mainly plans

- Track finding modules development has only started
  - plan to import the ATLAS seeding code at first
    currently reviewing the ATLAS track seeding code
    there is some level of caching ongoing (strategy for this developed)
    there is also room for improvement
  - future plan to revive parallel track finding approaches

- Track following, candidate classification, etc., room for a lot of ideas!
Track fitting

- First Kalman fitter prototype is implemented
  - gain matrix formalism
  - hole finding on the fly (Extrapolator gives that for free)
  - common backbone with Gaussian Sum Filter, DAF planned
Configuration

- ACTS is framework agnostic and persistency agnostic
  - aim to have a standalone R&D platform
  - do not want to mix persistency issues with data model structure

- Configuration encapsulated in configuration struct objects

```cpp
namespace Acts {
    /// doxygen documentation
    class WorkHorse {
        /// @struct Config for To
    struct Config {
        float coatColour; ///< configure the coat colour
        float maxPath;  ///< set the max path this horse can run
    };
};
}
```

- Seamless integration in experiment frameworks is a MUST
  - integration to Gaudi-Athena as an example achieved
  - provide a pre-loading of screen logging infrastructure
Concurrency - strategy

- Write non-mutable, const-correct code
  
  - Remove every use of "mutable" in ACTS

- Visitor cache objects to allow for stateless engines (that may cache)
  
  - caller creates a cache which stays local to the thread

```cpp
namespace Acts {
  /// doxygen documentation
  class WorkHorse {
    /// @struct Config for To
    struct Cache {
      float accumulatedPath; ///< configure the coat colour
    };
    /// method to make the horse run
    /// @param hCache - cache tracker for this horse
    /// @param coords - place where the horse should run to
    /// @return a result, horse may drop dead if max path is reached
    const RunResult run(Cache& hCache, const Vector3D& coords) const;
  };
}
```
Concurrency - status

- ACTS mini test framework allows for concurrent event processing
  - based on OpenMP, used for development & testing
  - number of threads can be set

- Predictive extrapolation example
  - FATRAS fast simulation without material effects
  - testing the load of up to 64 threads
  - shared geometry and magnetic field

CPU Usage Histogram
This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU usage value.
Vectorised code

- ACTS philosophy is to allow for plug/play mechanism
  - data structures should not stop/prevent vectorisation

```c
for(int i = 0; i < 421; i++) {
    double dR = 6P[i];
    double* dA = 6P[i+3];
    double dA0 = H0[2]*dA[1]-H0[1]*dA[2];
    double dB0 = H0[0]*dA[2]-H0[2]*dA[0];
    double dC0 = H0[1]*dA[0]-H0[0]*dA[1];
    if(i>=35) {dA0+=A0; dB0+=B0; dC0+=C0;}
    double dA2 = dA0+dA[0];
    double dA3 = dA0+dA[1];
    double dA2 = dA0+dA[2];
    double dA3 = dA0+dA[3];
    if(i>=35) {dA3+=A3-A00; dB3+=B3-A11; dC3+=C3-A22;}
    double dA4 = dA0+dA[4];
    double dA5 = dA0+dA[5];
    double dA6 = dA0+dA[6];
    if(i>=35) {dA6+=A6-A06; dB6+=B6-C6; dC6+=C6-C6;}
}
```

```
for(int i = 0; i < 42; i++){
    __m256 dR =_mm256_loadu_pd(&P[i]);
    __m256 dA =_mm256_loadu_pd(&P[i+3]);
    __m256 dA2 = __mm256_shuffle_pd(dA, dA, 1);
    __m256 dA3 = __mm256_shuffle_pd(dA, dA, 2);
    __m256 dA2 = __mm256_shuffle_pd(dA, dA, 0);
    __m256 dA3 = __mm256_shuffle_pd(dA, dA, 3);
    __m256 dA0 = __mm256_shuffle_pd(dA, dA, 3);
    __m256 dB0 = __mm256_shuffle_pd(dA, dA, 0);
    __m256 dB0 = __mm256_shuffle_pd(dA, dA, 1);
    __m256 dB0 = __mm256_shuffle_pd(dA, dA, 2);
    __m256 dB0 = __mm256_shuffle_pd(dA, dA, 3);
    if(i>=35) {dA0+=A0; dB0+=B0; dC0+=C0;}
    double dA2 = dA0+dA[0];
    double dA3 = dA0+dA[1];
    double dA2 = dA0+dA[2];
    double dA3 = dA0+dA[3];
    if(i>=35) {dA3+=A3-A00; dB3+=B3-A11; dC3+=C3-A22;}
    double dA4 = dA0+dA[4];
    double dA5 = dA0+dA[5];
    double dA6 = dA0+dA[6];
    if(i>=35) {dA6+=A6-A06; dB6+=B6-C6; dC6+=C6-C6;}
}
```

- Runge-Kutta integration but up to 2.4x faster using SIMD instructions
- Future CPUs are expected to further extend SIMD capacities

Roberto Agostino Vitillo
Framework integration

- ACTS integration in Gaudi for FCC-hh ongoing
  - ACTS integration for Athena/Gaudi will follow along the way
  - working on a generic GaudiWrapper for Tools/Algorithms
    configuration forwarded from Gaudi `declareProperty` to ACTS config structs

- Persistency mechanism is not part of ACTS
  - currently use ROOT, csv plugins for disk writing
  - could provide data streamers though if needed

- Several build and run-time options available
  - `Identifier` class can be overwritten with the experiment identifier class
  - track parameterisation can be chosen (though derivates, jacobins need to be provided)
  - logging facility can be pre-loaded (in current master, not in 0.03.00)
Conclusions

- ACTS has quite matured during the last year
  - public gitlab project (Core, Test-FW, fast track simulation in preparation)
  - increased functionality, steadily growing

- Toolset for geometry building from different backends
  - supporting different (so far cylindrical) detector geometries
    non-cylindrical detectors can be built, though no automated builders yet available
  - automated material mapping included from Geant4

- Dedicated care on Workflow and code integration
  - continuous integration testing, unit testing
  - documentation and code-style

- Keep tuned
Workflow

- git-based workflow
Example for TGeoDetectorElement

class TGeoDetectorElement : public DetectorElementBase
{
public:

/** Constructor */
TGeoDetectorElement(const Identifier& identifier,
TGeoNode* tGeoDetElement,
std::shared_ptr<const Acts::Transform3D> motherTransform
= nullptr);

/** Identifier */
virtual Identifier identify() const override;

/** Return local to global transform associated with this identifier */
virtual const Transform3D& transform(const Identifier& identifier = Identifier()) const override;

/** Return surface associated with this identifier, which should come from the */
virtual const Surface& surface(const Identifier& identifier = Identifier()) const override;

DD4hep

ACTS – Fast simulation